ENABLING A LIGHTWEIGHT SOFTWARE AGENT FRAMEWORK FOR SECURE AGENT-BASED ELECTRONIC COMMERCE APPLICATIONS

JORIS CLAESSENS BART PRENEEL JOOS VANDEWALLE
K.U.Leuven ESAT-COSIC
Kardinaal Mercierlaan 94, 3001 Heverlee, Belgium
Tel: +32-16-321853 – Fax: +32-16-321969
joris.claessens@esat.kuleuven.ac.be
http://www.esat.kuleuven.ac.be/~joclaess/

Although electronic commerce is a relatively new concept, it has already become a normal aspect of our daily life. The software agent technology is also relatively new. In the area of electronic commerce, software agents could be used for example to search the lowest prices and the best services, to buy goods on behalf of a user, etc. These applications involve a number of security issues that have to be solved. More in particular, communication security, system security, and application security, have to be provided. This paper describes how communication security is added to a lightweight agent framework. Secure agent-based electronic commerce applications require communication security services. As such, the addition of these services to the lightweight agent framework is an important enabler for the framework in order to be used for these applications.

1 Introduction

Money is a very important aspect of our modern society. We are confronted with several kinds of money in our daily life. Besides ordinary cash, electronic payment systems are nowadays frequently used. On the Internet, commerce itself is done electronically as well. People can buy books, compact disks, and even cars; flights can be booked at several airlines; music can be bought on-line directly from a producer, and downloaded in digital format; etc.

At the moment most electronic transactions are done manually, i.e., by going to the particular web site, selecting what is wanted, and entering a credit card number (if this payment mechanism is used) in order to pay. However, the concept of software agents can be used for electronic commerce to automate these tasks. Software agents will help people for example to find the lowest prices and the best services. They will also be able to pay without user intervention. Of course, as these applications involve electronic payments, and other sensitive operations and information, a number of security issues have to be solved first.

The work described in this paper intends to be a contribution in this area: a lightweight agent framework is secured, making it (partially) suitable for secure agent-based electronic commerce applications. In the next section, the security aspects with respect to agent-based electronic commerce applications will be briefly discussed. Thereafter, we will present the unsecured lightweight agent framework, and the mechanisms with which it is secured. How this is done exactly is explained in section 4. Section 5 discusses what has been achieved, and what should be done in the future. Finally, related work is described in section 6.

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.

2 Secure agent-based electronic commerce applications

Without giving a formal definition, a software agent is a very generic term for a piece of software that can operate autonomously, and that helps facilitate a human's task. Software agents can communicate, they often have learning capabilities, and they can sometimes travel from one computer to another computer (they are called mobile agents in that case). When agents are used in electronic commerce, a number of security aspects arise. These aspects are communication security, system security, and application security.

2.1 Communication security

A first important security aspect is communication security. The messages exchanged between software agents can be confidential, or can contain sensitive information. Software agents should be able to detect whether the messages are tampered with. Software agents should be able to verify who they are communicating with, and that the messages they receive are really originating from that entity. A communication infrastructure of an agent framework should therefore provide the following services: data confidentiality, entity authentication, and data authentication.

2.2 System security

Especially for mobile agents, system security is a second important security aspect. When agents can travel from host to host, hosts should be protected from malicious agents. On the other hand, agents should be protected from malicious hosts. Finally, agents should be protected from other agents.

2.3 Application security

In the scope of this paper, the communication security aspect only involves basic security services. These services are not enough in most electronic commerce applications. For electronic commerce, security services like non-repudiation and access control, are required as well. For example, non-repudiation ensures that agents (or their users) cannot deny having agreed upon an electronic contract; access control is applied when information is not available to everyone.

3 Towards secure agent communication

In this section, we present the lightweight agent framework to which we added communication security. This agent framework is Java-based. To some extent, Java offers interesting properties with respect to system security. Communication security is based on the SSL/TLS protocol, and is implemented using an SSL/TLS Java library.

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.

3.1 Lightweight agent framework

A Java-based agent framework for multi-agent applications, has been designed and implemented at the University of Linköping, Sweden [6]. A minimalistic approach has been used. The design goal was a small and simple framework that is easy to learn and to use, and at the same time generic and extensible. The agent framework provides agent communication and agent autonomy facilities to developers of agent applications.

There are two different kinds of agents within the framework. An *Agent* can send and receive messages, and it can act autonomously. A *System Agent* manages a system of agents. It keeps track of the agents in the system, handles communication, and provides support for agent autonomy.

The PingServer application is a very basic example, included in the package, that shows how to use the framework. The part of the code, important for agent communication, is given here. The same code will be extended in the next sections to show where exactly security is added into the framework, and how it affects an agent application.

PingServer.java:

```
SystemAgent system = new SystemAgent(4042);
Agent agent = new PingServerAgent();
system.addAgent(agent);
```

SystemAgent.java:

```
MessageServer server = new MessageServerSocketImp(port);
```

MessageServerSocketImp.java:

```
ServerSocket serverSocket = new ServerSocket(port);
Socket socket = serverSocket.accept();
```

First, a System Agent has to be setup, that will send and receive messages on a specific port (in this case 4042). Then, the actual Agent application is started, and registered with the System Agent. The PingServerAgent does nothing more than sending back the messages it receives. The MessageServer class handles the communication between the agents. Different implementations can be used. The MessageServerSocketImp is used, which performs the communication via standard TCP sockets.

3.2 Java

Java is a platform-independent object-oriented programming language developed by Sun. Java code runs in a virtual machine. The virtual machine on its turn, has to be implemented using platform-specific code, or can be directly provided by hardware (e.g., Java card and ring). Java is rather easy to learn and to work with, and this is probably one of the reasons for the fact that the lightweight agent framework is easy to learn and to use.

An important aspect of the language is the sandbox model. Applets (pieces of

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.

Java code that run within a browser environment) run inside a sandbox, and have limited privileges. They cannot for example read/write files on the local harddisk, and they cannot open network connections to other machines than the one they are originally downloaded from. In Java 2, every piece of code runs in a sandbox, and the borders of this sandbox – the security restrictions for that piece of code – can be defined differently based on who digitally signed the code. It is also impossible for a Java application to read the contents of other memory locations than the ones reserved for the application itself. This sandbox model will play an important role in the system security issues with respect to software agents.

A second important aspect is the provision of cryptographic functionality through the Java Cryptography Extension. These cryptographic services are of course needed for solving the communication security issues in a software agent framework.

3.3 SSL/TLS

Secure Sockets Layer (SSL) is an end-to-end security protocol, that provides entity authentication, data authentication, and data confidentiality, at the 'socket' level. It was an initiative of Netscape Communications. There are two versions of the protocol: 2.0 and 3.0. SSL 2.0 contains a number of flaws which are solved in SSL 3.0. The SSL 3.0 protocol was adopted by the IETF Transport Layer Security (TLS) working group, in which it is now standardized as the TLS 1.0 protocol [1] (some minor modifications compared to SSL 3.0, were made).

The Handshake protocol and the Record layer protocol are the two most important parts of the SSL/TLS protocol. The Record layer protocol provides the communication security services: data confidentiality by encrypting the messages, and data authentication by adding a MAC (Message Authentication Code). In addition, it is foreseen that the Record layer provides data compression too, although there is not yet a compression algorithm defined. Cryptographic keys are needed for encryption and calculating MACs. These keys, and other parameters (including the algorithms to be used) are negotiated during the Handshake protocol. The two communicating entities are authenticated as well in this phase.

IAIK-iSaSiLk [4] is a Java implementation of the SSL version 3.0 protocol. It operates on top of the IAIK-JCE Java Cryptography Extension APIs. Ideally, when securing an application using iSaSiLk, the only necessary modification of the existing Java code, is replacing all occurances of Socket by SSLSocket. Additional code has to be added for setting up the security parameters of the SSL'ed socket (a so-called security context). Note that IAIK-iSaSiLk and IAIK-JCE can be freely used during a short trial period, but that normal use requires a software license.

4 Securing the lightweight agent framework

The communication between agents in the lighweight agent framework can be socket based. As SSL offers the necessary services for securing this communication, and as the framework is implemented in Java, the IAIK-iSaSiLk package was used to add security to the framework implementation.

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.

4.1 Transparent security

In a first approach, security was added completely transparent for the actual agent application. The MessageServerSocketImp implementation was replaced by MessageServerSSLSocketImp. Within MessageServerSSLSocketImp, security is setup. All parameters are stored in one context class (different classes for server and client contexts), which has to be passed when opening a secured socket.

PingServer.java:

```
SystemAgent system = new SystemAgent(4042);
Agent agent = new PingServerAgent();
system.addAgent(agent);
```

SystemAgent.java:

```
MessageServer server = new MessageServerSSLSocketImp(port);
```

MessageServerSSLSocketImp.java:

```
SSLServerContext serverContext = new SSLServerContext();
SSLServerSocket serverSocket =
      new SSLServerSocket(port,serverContext);
SSLSocket socket = serverSocket.accept();
```

Security is as such completely transparent for the agent application. Note that the PingServer agent application did not need to be changed. When using SSL/TLS, this might be an ideal situation. However, the agent application cannot setup the security parameters, and has to use the ones offered by the system.

4.2 Security interface towards application agents

In the next approach, we provide the agent application with a basic interface to setup the security parameters. The parameters are setup using the iSaSiLk methods, and stored in a server or client context. This context is passed when setting up the System Agent, and the communication. Note that either the server context or the client context is passed when constructing the System Agent, depending on the agent being a server or a client application. The other context variable remains null.

SecurityAwarePingServer.java:

```
SSLServerContext serverContext = newSSLServerContext();
serverContext.setEnabledCipherSuites(cs);
serverContext.setRSACertificate(chain,
      SSLKeyStore.getPrivateKey(0,0));
serverContext.setTrustDecider(trustDecider);
SystemAgent system =
      new SystemAgent(4042,serverContext,null);
Agent agent = new PingServerAgent();
system.addAgent(agent);
```

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.

SystemAgent.java:

```
MessageServer server = new MessageServerAgentAware-
      SSLSocketImp(port,serverContext,clientContext);
```

MessageServerAgentAwareSSLSocketImp.java:

```
SSLServerSocket serverSocket =
      new SSLServerSocket(port,serverContext);
SSLSocket socket = serverSocket.accept();
```

As in the previous approach, all agents depending on the same system agent, use the same SSL tunnel, with the same security parameters. This is not surprising as also in the original unsecured framework, there is only one socket connection per system agent, shared by all agents that depend on this system agent.

4.3 Securing the messages itself

When applying security at the socket level of the framework, it is not possible to distinguish between agents that are subscribed to the same system agent. Suppose that agents cannot use their own personal system agent, and that they therefore have to use the same system agent. Suppose also that they need to work with different security parameters. Instead of securing the communication channel that has been setup by the system agent, the messages themselves should be secured in that case. A GSS-API [7] like approach could be used for that purpose. We did however not implement this alternative. Note that a similar approach has been done in [3] by defining security extensions for the Knowledge Query and Manipulation Language (KQML) [2], a language used in some frameworks for constructing agent messages.

4.4 Further comments

Setting up the server and/or client security context requires a lot of time. Having to perform an SSL handshake when establishing a connection, also causes overhead. There is thus certainly a decrease in performance compared to the original unsecured framework. However, the connections between the system agents remain open (for a certain period of time), so the decrease only occurs in the beginning. On a normal workstation or personal computer, encrypting and protecting the authenticity of the messages does not cause a noticeable performance decrease.

SSL/TLS is considered as an application-independent protocol, situated at the transport layer. In many SSL/TLS enabled applications, this is not clear. For example, some browsers and web servers implement SSL/TLS as part of the application. Within our secured framework, the real applications are situated on top of the agent 'transport layer', and are not aware of SSL/TLS (except for setting up the security context, which can be omitted when using the completely transparent approach).

Server and client system agents need to access their private key (for SSL/TLS key exchange and entity authentication). In a typical environment, these keys are stored on the local disk, encrypted with a symmetric key derived from a pass phrase

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.

(in a more advanced setup, smart cards would be used). When agents are manually started up, users can type in the pass phrase to unlock the key. When agents are automatically started up (e.g., by other agents), the keys either have to be stored in cleartext, or the pass phrase has to be hard coded in the agent software. It is clear that there is a link between communication security and system security. Moreover, when agents are mobile, and they want to setup a connection back to the originating host, they need a private key. In the completely transparent approach, one set of keys per host could be used, as part of the infrastructure. One should prevent that a malicious mobile agent can steal the private key, and use it on another host (e.g., keys only readable for the infrastructure classes). In combination with the completely transparent approach, a separate setup framework could be exploited for the keys, instead of hard coding keys in the infrastructure.

It is important to check whether the primary goals of the framework (simplicity, small code size, ...) are still maintained in the secured framework. Although the secured framework is still rather simple and easy to learn and to use, the code size is certainly not as small anymore, especially not if one takes into account the complete SSL/TLS and cryptography libraries. However, the current code size is not optimized, and it is for example possible to exclude all unnecessary algorithms and utilities from these libraries.

5 Achievements

The work described in this paper focused on the communication security aspect. By adding the SSL/TLS protocol, the lightweight agent framework now provides three communication security services: (system) agent authentication, data authentication, and data confidentiality. As seen in section 2, two other important security aspects with respect to agent-based electronic commerce applications are system security and application security. As mobile agents are not possible within the lightweight framework at this moment, the system security aspect is of less importance. Note that when the framework would be extended with mobility functionality, a Java implementation already provides a first level of system security (as explained in 3.2). Especially for mobile agents, system security plays an important role in the communication security aspect as well. For example, agents will need cryptographic keys to communicate securely. How will these keys be protected when the agents are traveling to other hosts? As we focused on the agent framework, we did not work on agent applications, and as such we did not cope with application security. For the time being, this is left for the developers of agent applications. Note again that in the case of mobile agents, system security is important for application security, for the same reasons as it is important for communication security.

6 Related work

A number of alternative (Java-based) agent frameworks exist: Aglets [5], JATLite [8], to name a few of them. These frameworks all have their software agent specific features. However, just as with the original lightweight agent framework, security does not seem to be an important issue in most frameworks. System security has

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.

sometimes being thought of, but communication security is mostly absent.

A lot of work has been done in the area of mobile agents and security [9] in general. Especially with respect to system security, some interesting solutions were presented in the past.

7 Conclusion

An existing Java-based lightweight software agent framework has been (partially) enabled for secure electronic commerce applications. The framework provided agent communication and agent autonomy facilities to developers of agent applications. A Java implementation of the SSL/TLS protocol was used to secure the communication facility. Communication security is one important aspect in software agent security in general. Other important aspects are system security and application security. System security is important when mobile agents are involved. Using Java already provides a certain level of system security. The work in this paper focused on the framework level, and as such application security has to be built in by the developers of agent applications.

Acknowledgements

This work has been done during a short research visit at the Laboratory for Intelligent Information Systems (IISLAB), Linköping university, Sweden. Thanks to Prof. Nahid Shahmehri of IISLAB for inviting me. Thanks also to the F.W.O. for financially supporting this visit, and to the I.W.T. for my overall research grant. Finally thanks to Prof. Bart Preneel and Prof. Joos Vandewalle for guiding my Ph.D. work.

References

1. T. Dierks and C. Allen. The TLS Protocol Version 1.0, January 1999. RFC2246.
2. T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*, 1997.
3. Q. He and K. P. Sycara. Towards a Secure Agent Society. In *ACM AA'98 Workshop on Deception, Fraud and Trust in Agent Societies*, 1998.
4. IAIK. Java crypto software. http://jcewww.iaik.tu-graz.ac.at/.
5. IBM. Aglets. http://www.trl.ibm.co.jp/aglets/.
6. M. Kindborg, J. Åberg, and N. Shahmehri. A lightweight agent framework for interactive multi-agent applications. In *Proceedings of Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agents*, pages 123–142, 1999. http://www.ida.liu.se/~mikki/PAAM99/.
7. J. Linn. Generic Security Services Application Program Interface, September 1993. RFC1508.
8. Stanford University. JATLite. http://java.stanford.edu/.
9. G. Vigna, editor. *Mobile Agents and Security*. Springer-Verlag, 1998. LNCS 1419.

Appeared in *Proceedings of the Workshop on Agents in Electronic Commerce*
WAEC'99, Hong Kong, December 14, 1999, pp 151–158
First Asia-Pacific Conference on Intelligent Agent Technology
IAT'99, Hong Kong, December 14-17, 1999.